**成都信息工程学院计算机学院**

**课**

**程**

**实**

**验**

**报**

**告**

|  |  |
| --- | --- |
| **实验课程：** | **网络游戏编程技术** |
| **实验项目：** | **游戏同步** |
| **指导教师：** |  |
| **学生姓名:** |  |
| **学生学号：** |  |
| **班 级：** |  |
| **实验地点：** |  |
| **实验时间：** | **2024 年 12 月 4 日 16 点~ 17:40 点** |
| **实验成绩：** |  |

## 一【上机实验内容】

1. 实验三的配对成功后跳转到游戏场景；

2. 向服务器发送位置同步、转身同步、开火同步协议；

3. 在接收到服务器转发来的位置同步、转身同步、开火同步协议时，实现相应操作；

4. 实现hero2的血条跟随、hero与rocket的碰撞检测、血量同步；

## 二【上机实验步骤】（重点）

### 操作步骤

实现客户端的位置发送，转身和开火协议。

接收协议时，客户端执行操作。

实现hero2的血量跟随

编写服务端代码

### 关键代码

客户端代码：

using System;

using System.Collections;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Net.Sockets;

using System.Text;

using TMPro;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Sample : MonoBehaviour

{

// Start is called before the first frame update

public Socket clientSocket;

NetworkStream networkStream;

StreamWriter writer;

StreamReader reader;

string recvStr;

bool isConnected = false;

public string myUsername;

private Coroutine receiveCoroutine; // 用于存储接收协程

GameObject hero1, hero2;

int frameCount = 0;

// Start is called before the first frame update

void Start()

{

getSocket();

DontDestroyOnLoad(this.gameObject);

hero1 = GameObject.Find("Hero");

hero2 = GameObject.Find("Hero2");

hero2.GetComponent<PlayerControl2>().Flip();

UnityEngine.Debug.Log(hero1);

UnityEngine.Debug.Log(hero2);

UnityEngine.Debug.Log(myUsername);

}

private IEnumerator ReceiveData() // 场景三的ReceiveData

{

while (isConnected)

{

if (networkStream.DataAvailable)

{

try

{

recvStr = reader.ReadLine();

if (!string.IsNullOrEmpty(recvStr))

{

string[] args = recvStr.Split(' ');

if (args[0] != "Position" && args[0] != "roomList")

{

UnityEngine.Debug.Log("收到服务器消息: " + recvStr);

}

// 检查 args 数组的长度

if (args.Length > 0)

{

if (args[0] == "Position")

{

SetPosition(args[1], args[2]);

}

else if (args[0] == "Flip")

{

hero2.GetComponent<PlayerControl2>().Flip();

}

else if (args[0] == "Fire")

{

GameObject.Find("Hero2/Gun").GetComponent<Gun2>().Fire();

}

else if (args[0] == "Health")

{

hero1.GetComponent<PlayerHealth>().health = float.Parse(args[1]);

hero1.GetComponent<PlayerHealth>().UpdateHealthBar();

}

else if (args[0] == "over")

{

Destroy(gameObject);

SceneManager.LoadScene(1);

}

}

else

{

UnityEngine.Debug.LogWarning("接收到的消息为空或格式不正确: " + recvStr);

}

}

}

catch (Exception e)

{

UnityEngine.Debug.LogError("接收消息失败: " + e.Message);

isConnected = false;

}

}

yield return null;

}

}

void getSocket()

{

Room room = FindObjectOfType<Room>();

myUsername = room.myUsername;

UnityEngine.Debug.Log(room);

if (room != null)

{

clientSocket = room.clientSocket;

networkStream = new NetworkStream(clientSocket);

writer = new StreamWriter(networkStream, Encoding.UTF8);

reader = new StreamReader(networkStream, Encoding.UTF8);

isConnected = true;

room.StopReceivingData(); // 停止场景二的ReceiveData

receiveCoroutine = StartCoroutine(ReceiveData()); // 启动场景三的ReceiveData

}

else

{

UnityEngine.Debug.LogError("未找到场景一的 Netword\_u1 实例");

}

}

void SendPosition()

{

float x = hero1.transform.position.x;

float y = hero1.transform.position.y;

string sendStr = "Position " + x.ToString() + " " + y.ToString() + " ";

//UnityEngine.Debug.Log("向服务器发送消息" + sendStr);

writer.WriteLine(sendStr);

writer.Flush();

}

public void SendHealth(float health)

{

UnityEngine.Debug.Log("向服务器发送消息" + "Health " + health);

writer.WriteLine("Health " + health);

writer.Flush();

}

public void Filp()

{

writer.WriteLine("Flip ");

writer.Flush();

}

public void Fire()

{

writer.WriteLine("Fire ");

writer.Flush();

}

void SetPosition(string strX, string strY)

{

float x = -float.Parse(strX);

float y = float.Parse(strY);

Vector2 newPosition = new Vector2(x, y);

hero2.transform.position = newPosition;

}

public void gameOver()

{

writer.WriteLine("over ");

writer.Flush();

}

// Update is called once per frame

void Update()

{

if (frameCount % 5 == 0) SendPosition();

frameCount++;

frameCount %= 5;

}

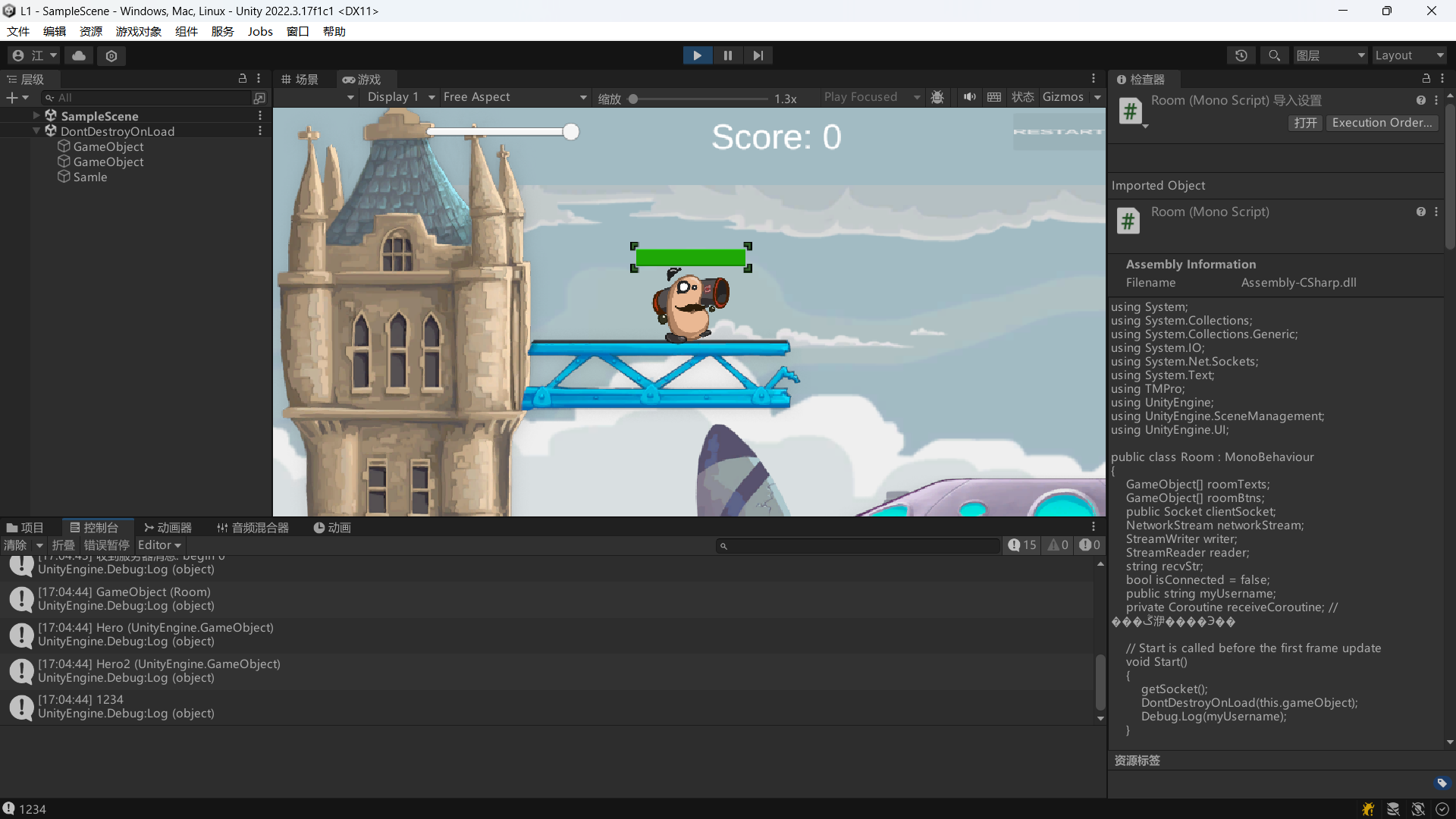
}

服务端代码：

package com.cy.socket;  
  
import com.cy.mapper.UserMapper;  
import com.cy.pojo.User;  
import com.cy.util.MD5Util;  
import com.cy.util.MybatisUtil;  
import org.apache.ibatis.session.SqlSession;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.HashMap;  
import java.util.HashSet;  
import java.util.Scanner;  
import java.util.Set;  
import java.util.concurrent.Executors;  
import java.util.concurrent.ScheduledExecutorService;  
import java.util.concurrent.TimeUnit;  
  
public class ChatServer {  
 private static final int *PORT* = 10002;  
 private static final Set<ClientHandler> *clientHandlers* = new HashSet<>();  
 private static SqlSession *sqlSession* = MybatisUtil.*sqlSessionFactory*();  
 private static UserMapper *userMapper* = *sqlSession*.getMapper(UserMapper.class);  
  
 private static HashMap<String, String> *map* = new HashMap<>();  
  
 public static void main(String[] args) {  
 *startServer*();  
 *listenForMessages*();  
 }  
  
 public static void startServer() {  
 System.*out*.println("服务器启动，监听端口：" + *PORT*);  
 new Thread(() -> {  
 try (ServerSocket serverSocket = new ServerSocket(*PORT*)) {  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 ClientHandler handler = new ClientHandler(clientSocket);  
 synchronized (*clientHandlers*) {  
 *clientHandlers*.add(handler);  
 System.*out*.println("客户端已添加，当前数量为" + *clientHandlers*.size());  
 }  
 handler.start();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }).start();  
 }  
  
 public static void listenForMessages() {  
 new Thread(() -> {  
 try (Scanner scanner = new Scanner(System.*in*)) {  
 while (true) {  
 String message = scanner.nextLine();  
 *sendMessageToAllClients*(message);  
 }  
 }  
 }).start();  
 }  
  
 public static void sendMessageToAllClients(String message) {  
 synchronized (*clientHandlers*) {  
 System.*out*.println("当前连接的客户端数量: " + *clientHandlers*.size());  
 for (ClientHandler handler : *clientHandlers*) {  
 handler.sendMessage(message);  
 // 打印客户端信息  
 System.*out*.println("消息发送到客户端: " + handler.getId());  
 }  
 }  
 }  
  
  
 private static class ClientHandler extends Thread {  
 private Socket socket;  
 private PrintWriter out;  
 private BufferedReader in;  
 private ScheduledExecutorService scheduler;  
  
 private StringBuilder messageToSend = new StringBuilder("roomList 0"); // 定义要发送的消息内容  
  
  
 public ClientHandler(Socket socket) {  
 this.socket = socket;  
 System.*out*.println("新客户端连接: " + socket.getInetAddress().getHostAddress() + ":" + socket.getPort());  
 try {  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new PrintWriter(socket.getOutputStream(), true);  
  
 // 初始化定时器  
 scheduler = Executors.*newScheduledThreadPool*(1);  
 for (ClientHandler handler : *clientHandlers*) {  
 if (!handler.messageToSend.equals("roomList 0")) {  
 this.messageToSend = handler.messageToSend;  
 }  
 }  
 scheduler.scheduleAtFixedRate(() -> sendMessage(messageToSend.toString()), 0, 2, TimeUnit.*SECONDS*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void run() {  
 try {  
 String message;  
  
 while ((message = in.readLine()) != null) {  
 if (!message.split(" ")[0].replace("\uFEFF", "").equals("Position")) {  
 System.*out*.println("收到消息: " + message);  
 }  
 String s1 = message.split(" ")[0];  
 //注册操作  
 if ("Regist".equals(s1.replace("\uFEFF", ""))) {  
 String password = MD5Util.*getMD5*(message.split(" ")[2].replace("\uFEFF", ""));  
 String username = message.split(" ")[1].replace("\uFEFF", "");  
 String ip = socket.getInetAddress().getHostAddress() + ":" + socket.getPort();  
 User user = *userMapper*.select(username);  
 //用户名存在  
 if (user != null) {  
 sendMessage("RegistFail 1 ");  
 System.*out*.println("用户名存在");  
 } else {  
 User user1 = new User();  
 user1.setUsername(username);  
 user1.setPassword(password);  
 user1.setOnline(0);  
 user1.setScore(0);  
 user1.setUserIp(ip);  
 *userMapper*.regist(user1);  
 *sqlSession*.commit();  
 sendMessage("RegistSuccess ");  
 System.*out*.println("注册成功");  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("Login")) {//登录操作  
 String password = MD5Util.*getMD5*(message.split(" ")[2].replace("\uFEFF", ""));  
 String username = message.split(" ")[1].replace("\uFEFF", "");  
 User user = *userMapper*.select(username);  
 if (user == null) {  
 sendMessage("LoginFail 1");  
 System.*out*.println("用户不存在");  
 } else if (!user.getPassword().equals(password)) {  
 sendMessage("LoginFail 2");  
 System.*out*.println("密码错误");  
 } else if (user.getOnline() == 1) {  
 sendMessage("LoginFail 3");  
 System.*out*.println("用户已登录");  
 } else {  
 String ip = socket.getInetAddress().getHostAddress() + ":" + socket.getPort();  
 *userMapper*.online(username, ip);  
 *sqlSession*.commit();  
 sendMessage("LoginSuccess");  
 System.*out*.println("登录成功");  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("createRoom")) {  
 String[] s = messageToSend.toString().split(" ");  
 StringBuilder ms = new StringBuilder("roomList " + (Integer.*parseInt*(s[1]) + 1) + " ");  
 String username = *userMapper*.findByIp(socket.getInetAddress().getHostAddress() + ":" + socket.getPort()).getUsername();  
 for (int i = 2; i < s.length; i++) {  
 ms.append(s[i]).append(" ");  
 }  
 ms.append(username).append(" ");  
 messageToSend = ms;  
 broadcast(ms.toString());  
 updateRooms(ms.toString());  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("enterRoom")) {  
 String username = message.split(" ")[1];  
 User select = *userMapper*.select(username);  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(select.getUserIp())) {  
 handler.sendMessage("begin 1");  
 }  
 }  
 sendMessage("begin 0");  
 *map*.put(select.getUserIp(), this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 *map*.put(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort(), select.getUserIp());  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("Position")) {  
 String ip = *map*.get(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(ip)) {  
 handler.sendMessage("Position " + message.split(" ")[1].replace("\uFEFF", "") + " " + message.split(" ")[2].replace("\uFEFF", "") + " ");  
 }  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("File")) {  
 String ip = *map*.get(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(ip)) {  
 handler.sendMessage("Flip ");  
 }  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("Fire")) {  
 String ip = *map*.get(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(ip)) {  
 handler.sendMessage("Fire ");  
 }  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("Health")) {  
 String ip = *map*.get(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(ip)) {  
 handler.sendMessage("Health " + message.split(" ")[1].replace("\uFEFF", "") + " ");  
 }  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("over")) {  
 String username = *userMapper*.findByIp(socket.getInetAddress().getHostAddress() + ":" + socket.getPort()).getUsername();  
 System.*out*.println("游戏结束，赢家为：" + username);  
 String ip = *map*.get(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(ip)) {  
 handler.sendMessage("over ");  
 }  
 }  
 *map*.remove(this.socket.getInetAddress().getHostAddress() + ":" + this.socket.getPort());  
 *map*.remove(ip);  
 }  
 //这会把消息重新发送给所有客户端  
// broadcast(message);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 // 关闭定时器  
 if (scheduler != null) {  
 scheduler.shutdown();  
 }  
 synchronized (*clientHandlers*) {  
 *clientHandlers*.remove(this);  
 String ip = socket.getInetAddress().getHostAddress() + ":" + socket.getPort();  
 User user = *userMapper*.findByIp(ip);  
 if (user != null) {  
 *userMapper*.updateByIp(ip);  
 *sqlSession*.commit();  
 }  
 System.*out*.println("客户端数量为" + *clientHandlers*.size());  
 }  
 socket.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
  
 public void sendMessage(String message) {  
 out.println(message);  
 }  
  
 private void broadcast(String message) {  
 synchronized (*clientHandlers*) {  
 for (ClientHandler handler : *clientHandlers*) {  
 handler.sendMessage(message);  
 }  
 }  
 }  
  
 private void updateRooms(String message) {  
 synchronized (*clientHandlers*) {  
 for (ClientHandler handler : *clientHandlers*) {  
 if (!handler.messageToSend.equals(this.messageToSend)) {  
 handler.messageToSend = new StringBuilder(message);  
 }  
 }  
 }  
 }  
 }  
}

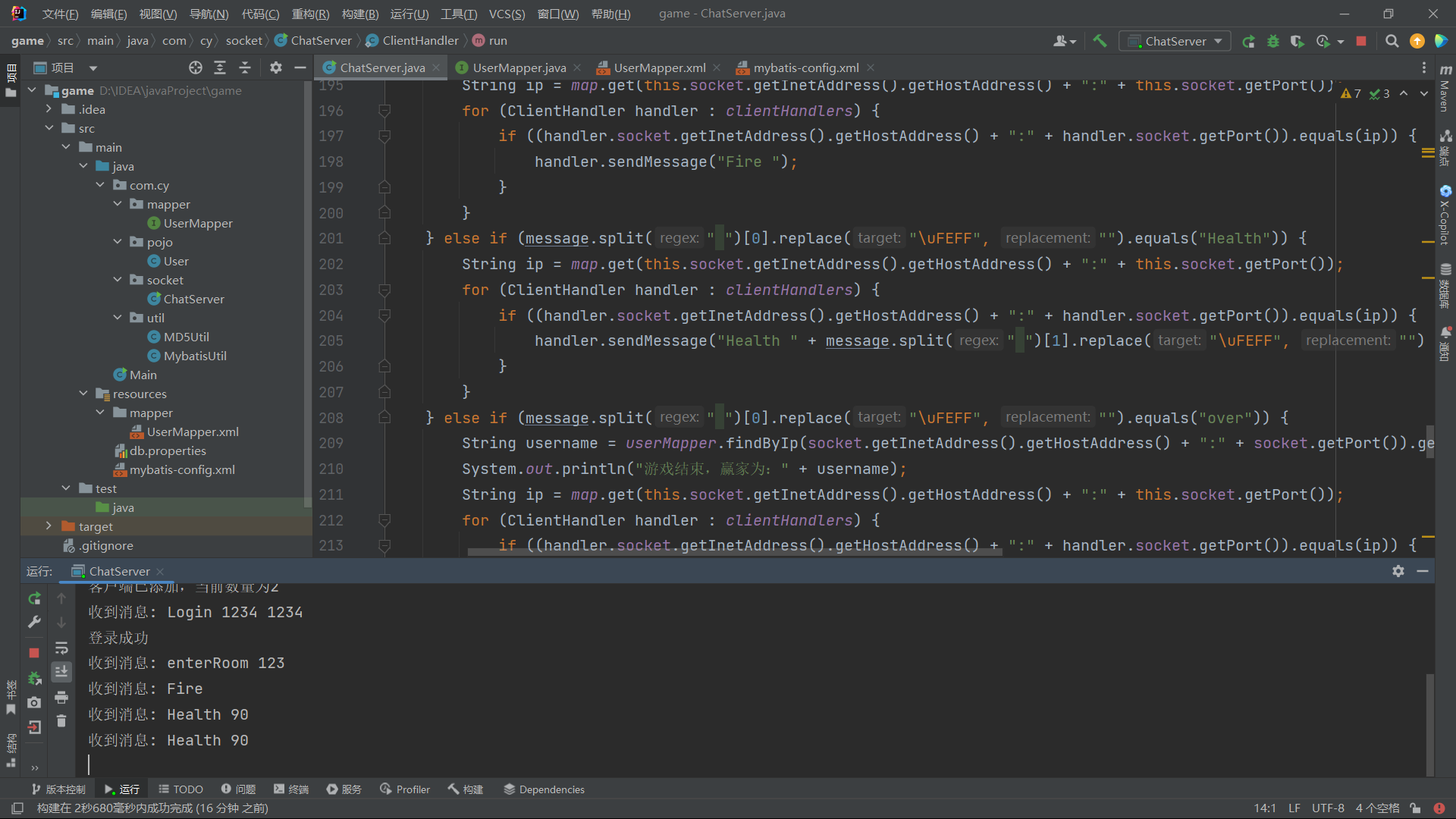
### 运行结果截图

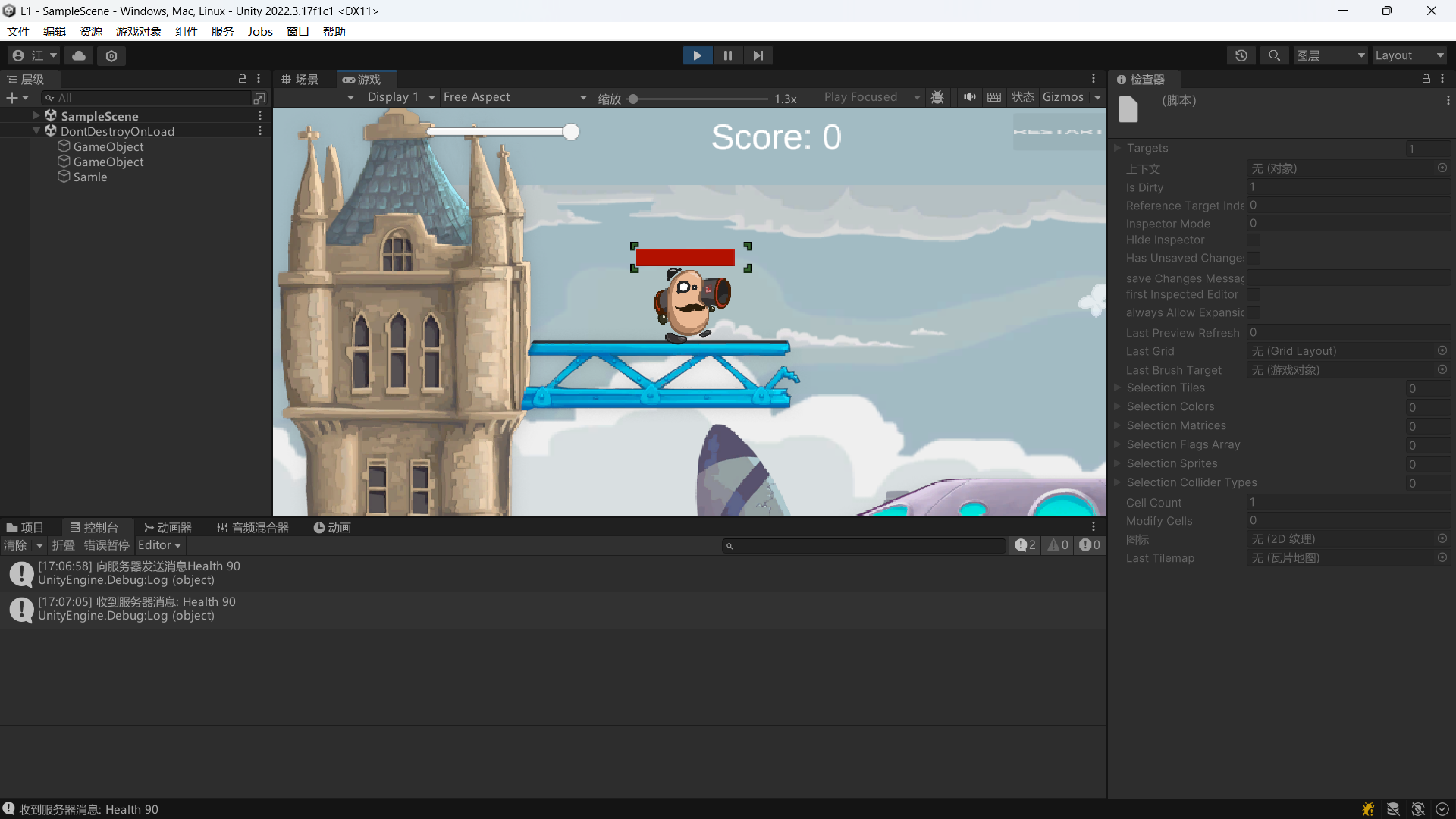
进入游戏场景：





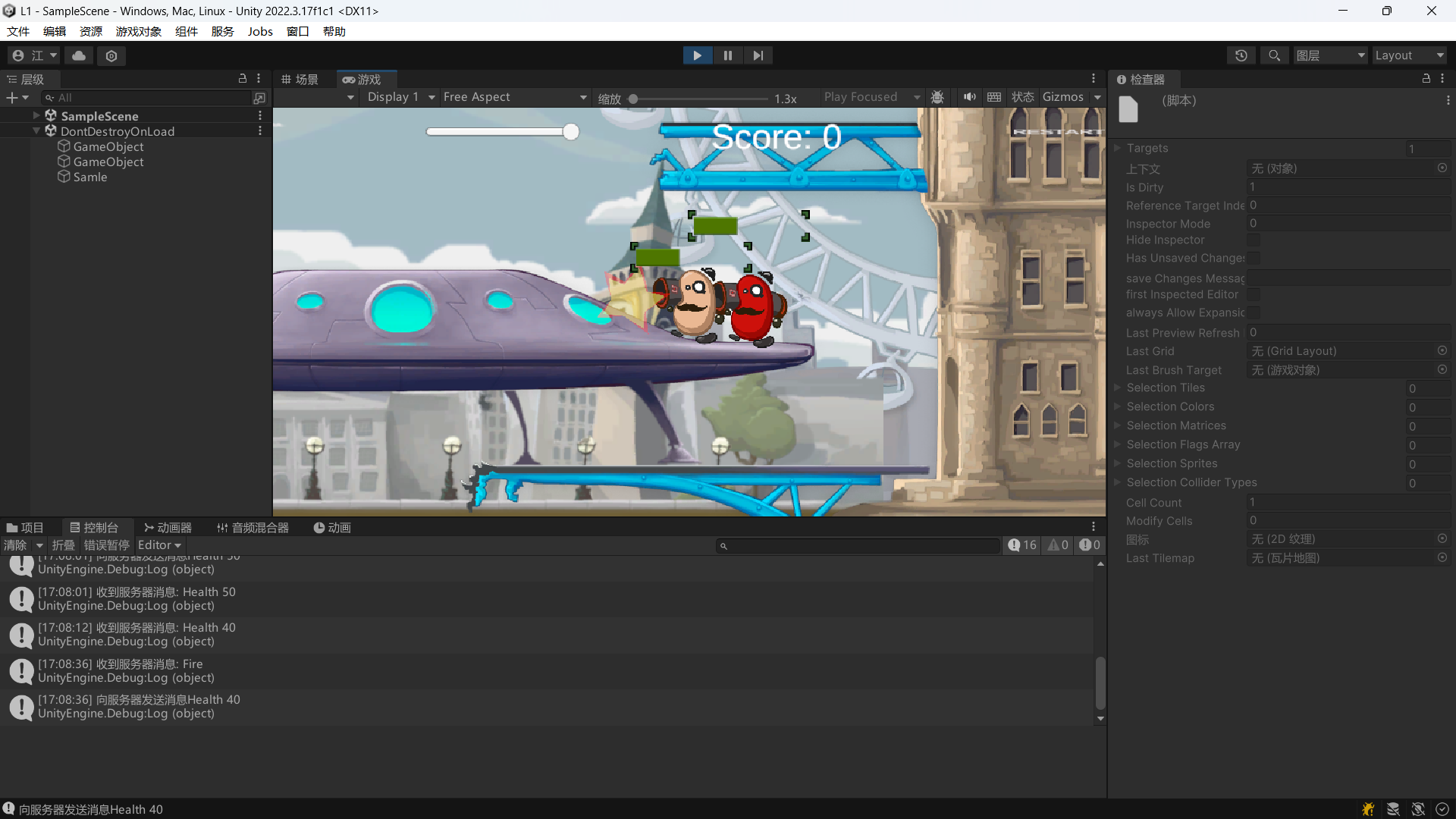
开火与血量同步：

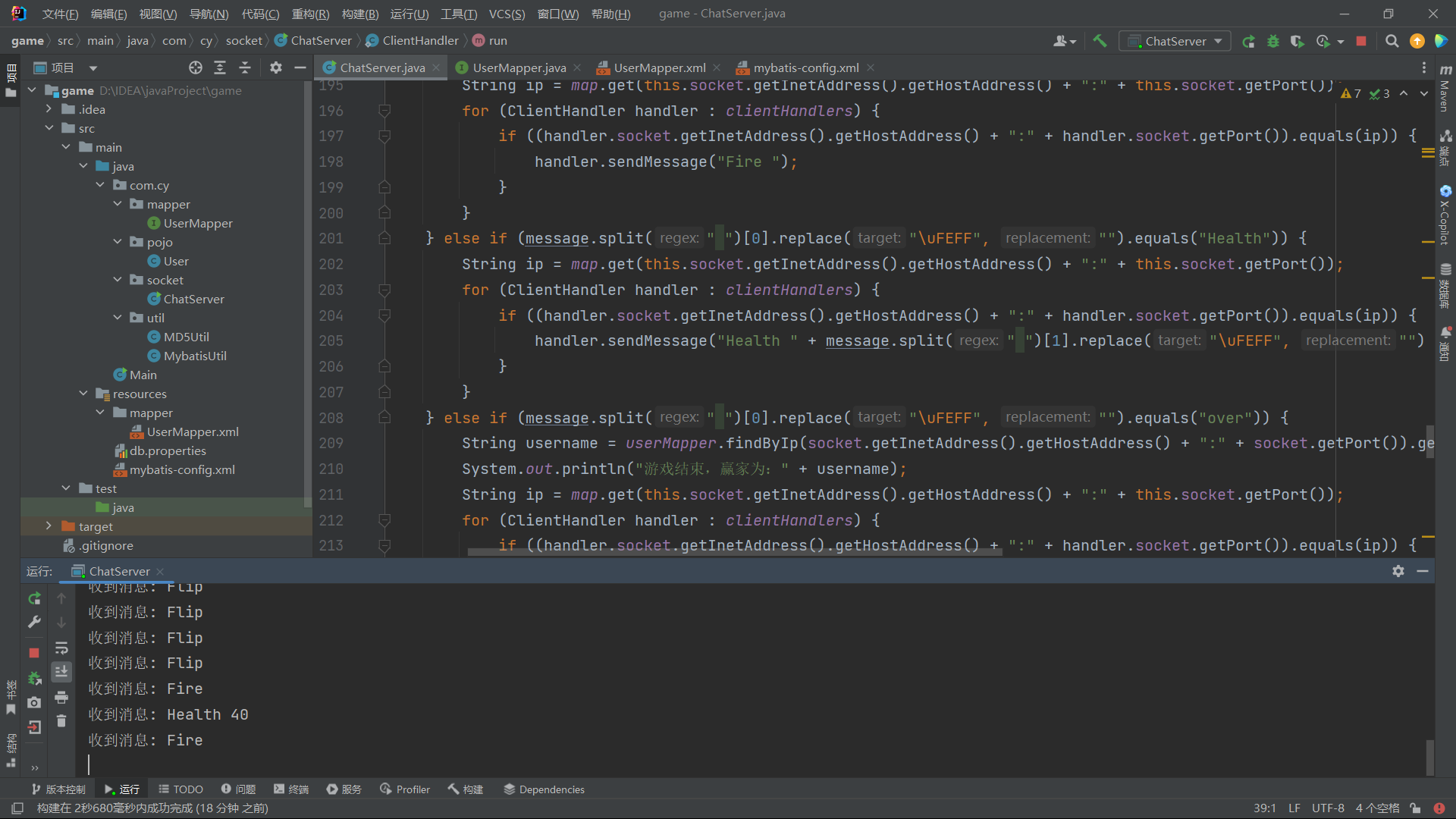




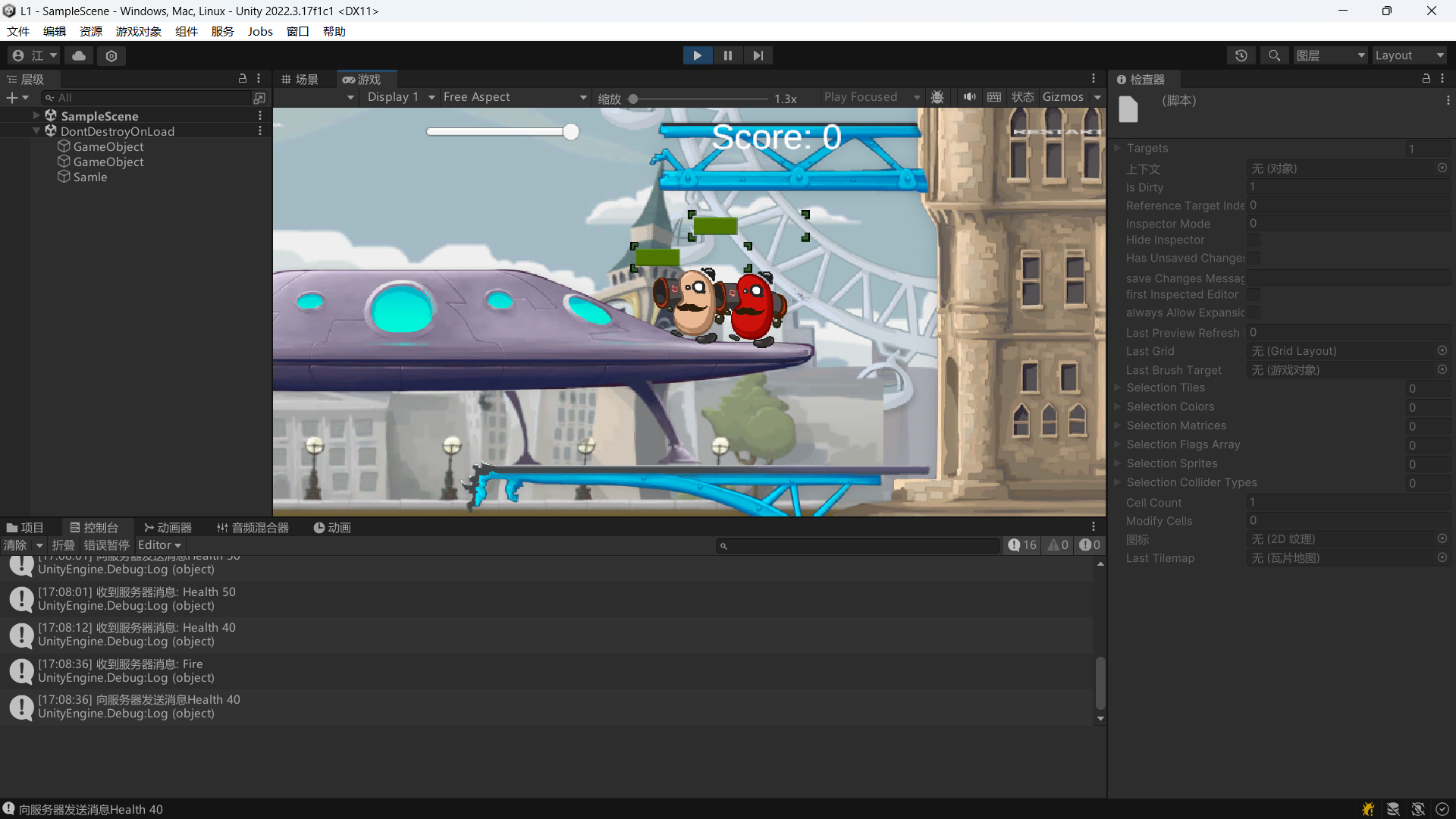
位置同步，转身同步：



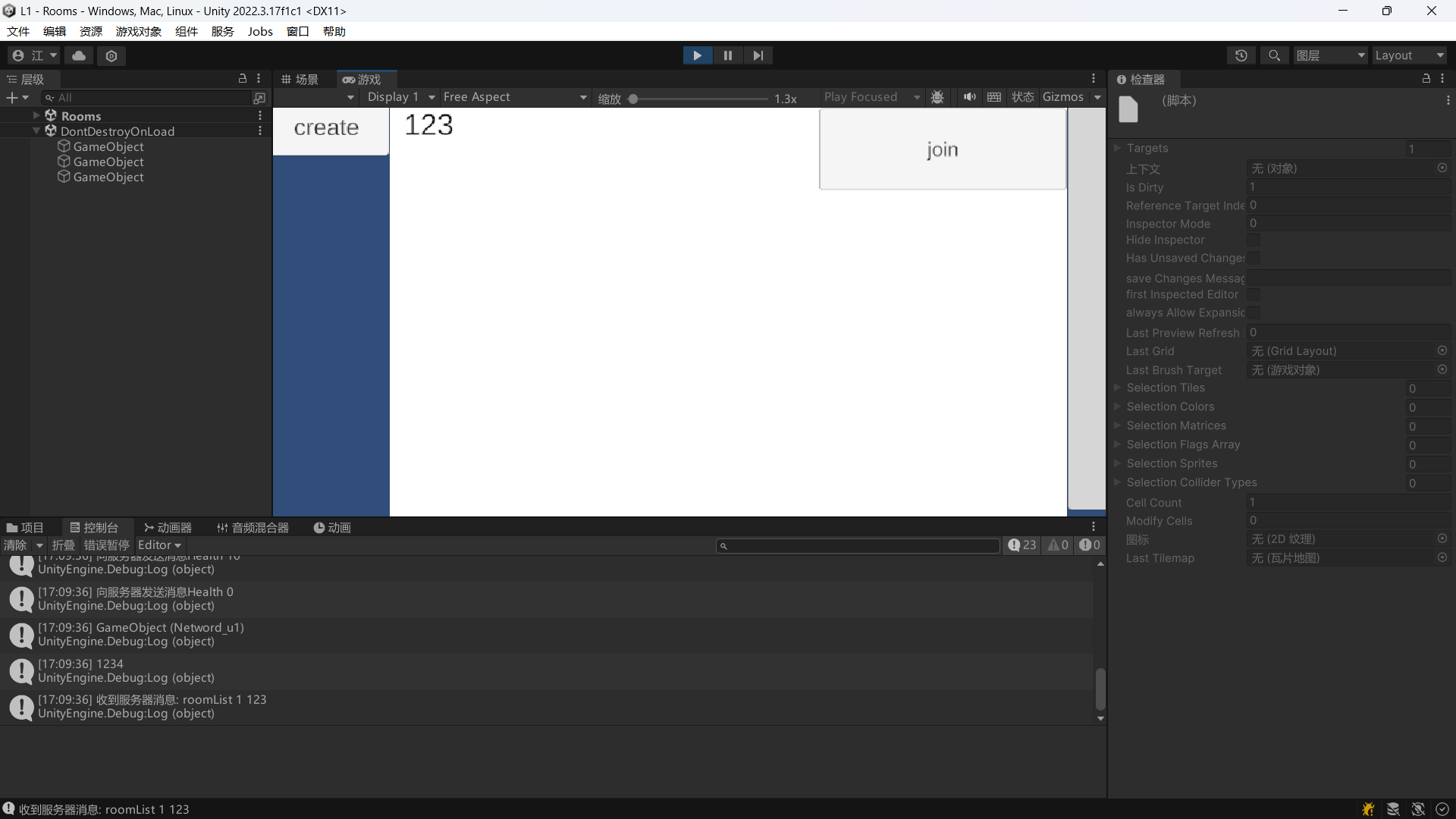


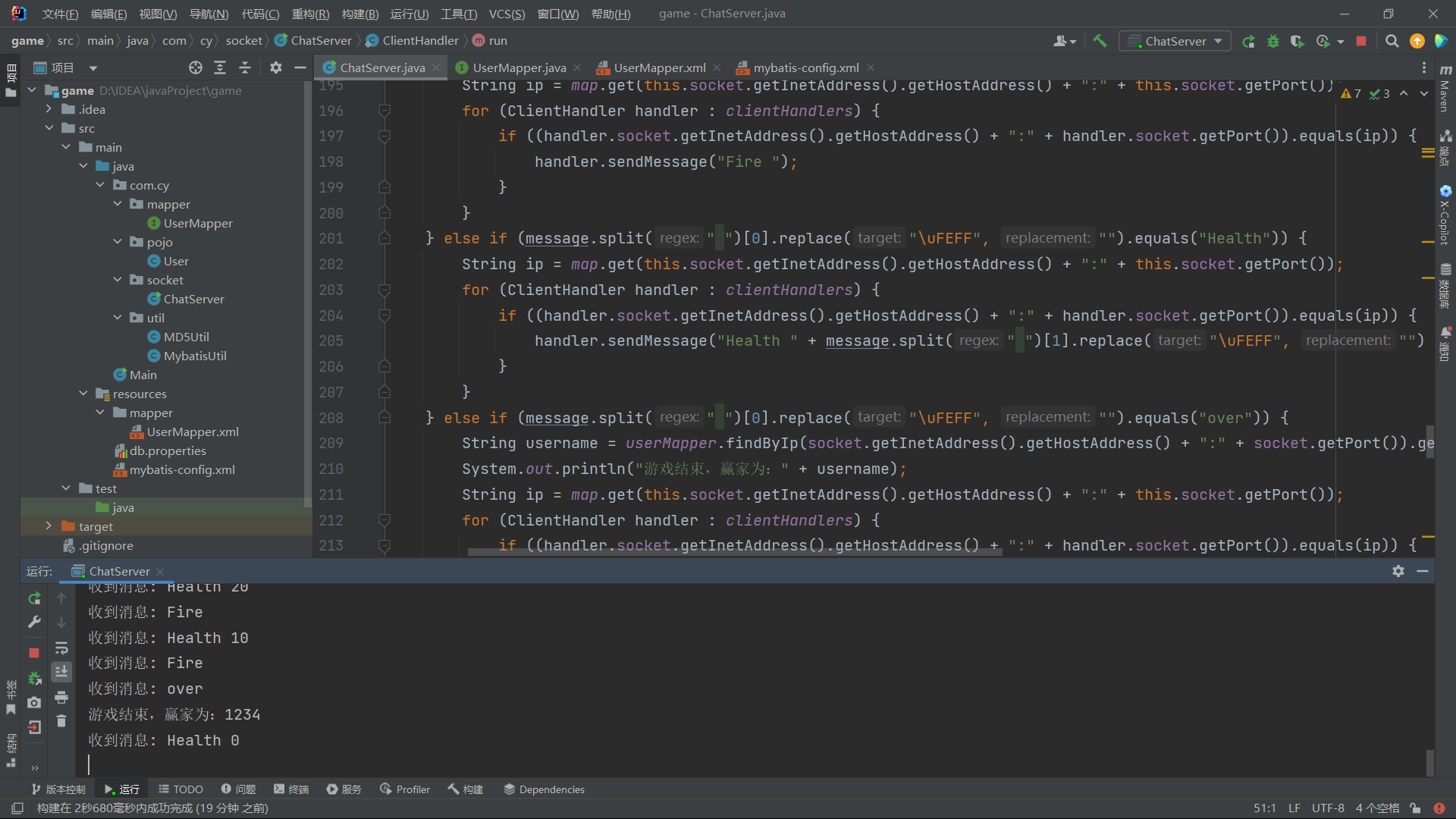


Hero2血量跟随：



游戏结束向服务端发送结果：





## 三【上机实验中的其他它问题及心得】

这次实验我学到了客户端与服务端位置同步，血量同步的编写。